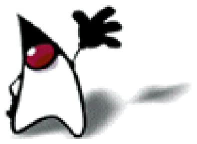
XML Schemas



XML Schemas

- “Schemas” is a general term--DTDs are a form of XML schemas
  - According to the dictionary, a schema is “a structured framework or plan”
- When we say “XML Schemas,” we usually mean the W3C XML Schema Language
  - This is also known as “XML Schema Definition” language, or XSD
  - I’ll use “XSD” frequently, because it’s short
- DTDs, XML Schemas, and RELAX NG are all XML schema languages

Why XML Schemas?

- DTDs provide a very weak specification language
  - You can’t put any restrictions on text content
  - You have very little control over mixed content (text plus elements)
  - You have little control over ordering of elements
- DTDs are written in a strange (non-XML) format
  - You need separate parsers for DTDs and XML
- The XML Schema Definition language solves these problems
  - XSD gives you much more control over structure and content
  - XSD is written in XML

Why not XML schemas?

- DTDs have been around longer than XSD
  - Therefore they are more widely used
  - Also, more tools support them
- XSD is very verbose, even by XML standards
- More advanced XML Schema instructions can be non-intuitive and confusing

- Nevertheless, XSD is not likely to go away quickly

Referring to a schema

- To refer to a DTD in an XML document, the reference goes *before* the root element:

  -
    ```
    <?xml version="1.0"?>
    <!DOCTYPE rootElement SYSTEM "url">
    <rootElement> ... </rootElement>
    ```
- To refer to an XML Schema in an XML document, the reference goes *in* the root element:

  -
    ```
    <?xml version="1.0"?>
    <rootElement
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                (The XML Schema Instance reference is required)
        xsi:noNamespaceSchemaLocation="url.xsd">
                (This is where your XML Schema definition can be found)
      ...
    </rootElement>
    ```

The XSD document

- Since the XSD is written in XML, it can get confusing which we are talking about
- Except for the additions to the root element of our XML data document, the rest of this lecture is about the XSD schema document
- The file extension is `.xsd`
- The root element is `<schema>`
- The XSD starts like this:

  -
    ```
    <?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.rg/2001/XMLSchema">
    ```

<schema>

- The <schema> element may have attributes:
  - xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - This is necessary to specify where all our XSD tags are defined
  - elementFormDefault="qualified"
    - This means that all XML elements must be qualified (use a namespace)
    - It is highly desirable to qualify all elements, or problems will arise when another schema is added

“Simple” and “complex” elements

- A “simple” element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn’t simple, it’s “complex”
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a simple element

- A simple element is defined as
  `<xs:element name="`***name***`" type="`***type***`" />`
  where:
  - ***name*** is the name of the element
  - the most common values for ***type*** are

xs:boolean	xs:integer
xs:date	xs:string
xs:decimal	xs:time

- Other attributes a simple element may have:
  - `default="`***default value***`"` *if no other value is specified*
  - `fixed="`***value***`"` *no other value may be specified*

Defining an attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as
  `<xs:attribute name="`***name***`" type="`***type***`" />`
  where:
  - ***name*** and ***type*** are the same as for `xs:element`
- Other attributes a simple element may have:

`default="`***default value***`"`	*if no other value is specified*
`fixed="`***value***`"`	*no other value may be specified*
`use="optional"`	*the attribute is not required (default)*
`use="required"`	*the attribute must be present*

Restrictions, or “facets”

- The general form for putting a restriction on a text value is:
  - `<xs:element name="`*`name`*`">` *(or* `xs:attribute`*)*

```
<xs:element  name="name">
    <xs:restriction base="type">
        ... the restrictions ...
    </xs:restriction>
</xs:element>
```

- For example:

```
<xs:element  name="age">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0">
        <xs:maxInclusive value="140">
    </xs:restriction>
</xs:element>
```

Restrictions on numbers

- minInclusive -- number must be ≥ the given ***value***
- minExclusive -- number must be > the given ***value***
- maxInclusive -- number must be ≤ the given ***value***
- maxExclusive -- number must be < the given ***value***
- totalDigits -- number must have exactly ***value*** digits
- fractionDigits -- number must have no more than ***value*** digits after the decimal point

Restrictions on strings

- length -- the string must contain exactly ***value*** characters
- minLength -- the string must contain at least ***value*** characters
- maxLength -- the string must contain no more than ***value*** characters
- pattern -- the ***value*** is a regular expression that the string must match
- whiteSpace -- not really a "restriction"--tells what to do with whitespace
  - value="preserve" Keep all whitespace
  - value="replace" Change all whitespace characters to spaces
  - value="collapse" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

  -
    ```
    <xs:element name="season">
        <xs:simpleType>
            <xs:restriction  base="xs:string">
                <xs:enumeration value="Spring"/>
                <xs:enumeration value="Summer"/>
                <xs:enumeration value="Autumn"/>
                <xs:enumeration value="Fall"/>
                <xs:enumeration value="Winter"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    ```

Complex elements

- A complex element is defined as

```
<xs:element  name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- Example:

```
<xs:element  name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- `<xs:sequence>` says that elements must occur in this order
- Remember that attributes are always simple types

Global and local definitions

- Elements declared at the "top level" of a <schema> are available for use throughout the schema
- Elements declared within a xs:complexType are local to that type
- Thus, in

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

  the elements firstName and lastName are only locally declared
- The order of declarations at the "top level" of a <schema> *do not* specify the order in the XML data document

Declaration and use

- So far we've been talking about how to *declare* types, not how to *use* them
- To *use* a type we have declared, use it as the value of `type="…"`
  - Examples:
    - `<xs:element name="student" type="person"/>`
    - `<xs:element name="professor" type="person"/>`
  - Scope is important: you cannot use a type if is local to some other type

xs:sequence

- We’ve already seen an example of a complex type whose elements must occur in a specific order:

-
  ```
  <xs:element   name="person">
     <xs:complexType>
       <xs:sequence>
         <xs:element  name="firstName"  type="xs:string" />
         <xs:element  name="lastName"  type="xs:string" />
       </xs:sequence>
     </xs:complexType>
   </xs:element>
  ```

xs:all

- xs:all allows elements to appear in any order

-
  ```
  <xs:element  name="person">
     <xs:complexType>
        <xs:all>
           <xs:element  name="firstName"  type="xs:string" />
           <xs:element  name="lastName"  type="xs:string" />
        </xs:all>
     </xs:complexType>
   </xs:element>
  ```
- Despite the name, the members of an xs:all group can occur once or not at all
- You can use minOccurs="0" to specify that an element is optional (default value is 1)
  - In this context, maxOccurs is always 1

Referencing

- Once you have defined an element or attribute (with name="..."), you can refer to it with ref="..."
- Example:

  -
    ```
    <xs:element  name="person">
       <xs:complexType>
          <xs:all>
             <xs:element  name="firstName"  type="xs:string" />
             <xs:element  name="lastName"  type="xs:string" />
          </xs:all>
       </xs:complexType>
     </xs:element>
    ```
  - `<xs:element  name="student"  ref="person">`
  - Or just: `<xs:element  ref="person">`

Text element with attributes

- If a text element has attributes, it is no longer a simple type

  -
    ```
    <xs:element name="population">
       <xs:complexType>
          <xs:simpleContent>
             <xs:extension base="xs:integer">
                <xs:attribute name="year”
                              type="xs:integer">
             </xs:extension>
          </xs:simpleContent>
       </xs:complexType>
    </xs:element>
    ```

Empty elements

- Empty elements are (ridiculously) complex

```
<xs:complexType  name="counter">
     <xs:complexContent>
          <xs:extension  base="xs:anyType"/>
          <xs:attribute  name="count"  type="xs:integer"/>
    </xs:complexContent>
</xs:complexType>
```

Mixed elements

- Mixed elements may contain both text and elements
- We add mixed="true" to the xs:complexType element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

-
  ```
  <xs:complexType  name="paragraph"  mixed="true">
      <xs:sequence>
          <xs:element  name="someName”
                       type="xs:anyType"/>
     </xs:sequence>
  </xs:complexType>
  ```

Extensions

- You can base a complex type on another complex type

-
  ```
  <xs:complexType  name="newType">
      <xs:complexContent>
          <xs:extension  base="otherType">
              ...new stuff...
          </xs:extension>
      </xs:complexContent>
  </xs:complexType>
  ```

Predefined string types

- Recall that a simple element is defined as:
  `<xs:element name="`***name***`" type="`***type***`" />`
- Here are a few of the possible string types:
  - xs:string -- a string
  - xs:normalizedString -- a string that doesn't contain tabs, newlines, or carriage returns
  - xs:token -- a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:
  - enumeration, length, maxLength, minLength, pattern, whiteSpace

Predefined date and time types

- xs:date -- A date in the format ***CCYY-MM-DD***, for example, 2002-11-05
- xs:time -- A date in the format ***hh:mm:ss*** (hours, minutes, seconds)
- xs:dateTime -- Format is ***CCYY-MM-DD*****T*****hh:mm:ss***
  - The T is part of the syntax
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined numeric types

- Here are some of the predefined numeric types:

xs:decimal	xs:positiveInteger
xs:byte	xs:negativeInteger
xs:short	xs:nonPositiveInteger
xs:int	xs:nonNegativeInteger
xs:long	

- Allowable restrictions on numeric types:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

The End